APPENDIX A

"EXPRESS MAIL" Mailing Label Number EI267842785US

Date of Deposit October 24, 1997

I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as **"Express Mail Post Office To Addressee"** with sufficient postage on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Tina Grimstead-Campbell

APPENDIX A

Card Class File Format For Preferred Embodiment

Introduction

The card class file is a compressed form of the original class file(s). The card class file contains only the semantic information required to interpret Java programs from the original class files. The indirect references in the original class file are replaced with direct references resulting in a compact representation. The card class file format is based on the following principles:

1. ***Stay close to the standard class file format***: The card class file format should remain as close to the standard class file format as possible. The Java byte codes in the class file remain unaltered. Not altering the byte codes ensures that the structural and static constraints on them remain verifiably intact.
2. ***Ease of implementation***: The card class file format should be simple enough to appeal to Java Virtual Machine implementers. It must allow for different yet behaviorally equivalent implementations.
3. ***Feasibility***: The card class file format must be compact in order to accommodate smart card technology. It must meet the constraints of today's technology while not losing sight of tomorrow's innovations.

This document is based on Chapter 4, "The class file format", in the book titled "The Java™ Virtual Machine Specification"[1], henceforth referred to as the Red book. Since the document is based on the standard class file format described in the Red book, we only present information that is different. The Red book serves as the final authority for any clarification.

The primary changes from the standard class file format are:

- The constant pool is optimized to contain only 16-bit identifiers and, where possible, indirection is replaced by a direct reference.
- Attributes in the original class file are eliminated or regrouped.

The Java Card class File Format

This section describes the Java Card class file format. Each card class file contains one or many Java types, where a type may be a class or an interface.

A card class file consists of a stream of 8-bit bytes. All 16-bit, 32-bit, and 64-bit quantities are constructed by reading in two, four, and eight consecutive 8-bit bytes, respectively. Multi-byte data items are always stored in big-endian order, where the high bytes come first. In Java, this format is supported by interfaces java.io.DataInput and java.io.DataOutput and classes such as java.io.DataInputStream and java.io.DataOutputStream.

We define and use the same set of data types representing Java class file data: The types u1, u2, and u4 represent an unsigned one-, two-, or four-byte quantity, respectively. In Java, these types may be read by methods such as readUnsignedByte, readUnsignedShort, and readInt of the interface java.io.DataInput.

The card class file format is presented using pseudo-structures written in a C-like structure notation. To avoid confusion with the fields of Java Card Virtual Machine classes and class instances, the contents of the structures describing the card class file format are referred to as items. Unlike the fields of a C structure, successive items are stored in the card class file sequentially, without padding or alignment.

Variable-sized tables, consisting of variable-sized items, are used in several class file structures. Although we will use C-like array syntax to refer to table items, the fact that tables are streams of varying-sized structures means that it is not possible to directly translate a table index into a byte offset into the table. Where we refer to a data structure as an array, it is literally an array.

In order to distinguish between the card class file structure and the standard class file structure, we add capitalization; for example, we rename field_info in the original class file to FieldInfo in the card class file.

Card Class File

A card class file contains a single CardClassFile structure:

```
CardClassFile {
    u1 major_version;
    u1 minor_version;
    u2 name_index;
    u2 const_size;
    u2 max_class;
    CpInfo constant_pool[const_size];
    ClassInfo class[max_class];
}
```

The items in the CardClassFile structure are as follows:

minor_version, major_version

The values of the minor_version and major_version items are the minor and major version numbers of the off-card Java Card Virtual Machine that produced this card class file. An implementation of the Java Card Virtual Machine normally supports card class files having a given major version number and minor version numbers 0 through some particular minor_version.

Only the Java Card Forum may define the meaning of card class file version numbers.

name_index

The value of the name_index item must represent a valid Java class name. The Java class name represented by name_index must be exactly the same Java class name that corresponds to the main application that is to run in the card. A card class file contains several classes or interfaces that constitute the application that runs in the card. Since Java allows each class to contain a main method there must be a way to distinguish the class file containing the main method which corresponds to the card application.

const_size

The value of const_size gives the number of entries in the card class file constant pool. A constant_pool index is considered valid if it is greater than or equal to zero and less than const_size.

max_class

This value refers to the number of classes present in the card class file. Since the name resolution and linking in the Java Card are done by the off-card Java Virtual Machine all the class files or classes required for an application are placed together in one card class file.

constant_pool[]

The constant_pool is a table of variable-length structures (0) representing various string constants, class names, field names, and other constants that are referred to within the CardClassFile structure and its substructures.

The first entry in the card class file is constant_pool[0].

Each of the constant_pool table entries at indices 0 through const_size is a variable-length structure (0).

class[]

The class is a table of max_class classes that constitute the application loaded onto the card.

Constant Pool

All constant_pool table entries have the following general format:

```
CpInfo {
    u1 tag;
    u1 info[];
}
```

Each item in the constant_pool table must begin with a 1-byte tag indicating the kind of cp_info entry. The contents of the info array varies with the value of tag. The valid tags and their values are the same as those specified in the Red book.

Each tag byte must be followed by two or more bytes giving information about the specific constant. The format of the additional information varies with the tag value. Currently the only tags that need to be included are CONSTANT_Class, CONSTANT_FieldRef, CONSTANT_MethodRef and CONSTANT_InterfaceRef. Support for other tags be added as they are included in the specification.

CONSTANT_Class

The CONSTANT_Class_info structure is used to represent a class or an interface:

```
CONSTANT_ClassInfo {
    u1 tag;
    u2 name_index;
}
```

The items of the CONSTANT_Class_info structure are the following:

tag

The tag item has the value CONSTANT_Class (7).

name_index

The value of the name_index item must represent a valid Java class name. The Java class name represented by name_index must be exactly the same Java class name that is described by the corresponding CONSTANT_Class entry in the constant_pool of the original class file.

CONSTANT_Fieldref, CONSTANT_Methodref, and CONSTANT_InterfaceMethodref

Fields, methods, and interface methods are represented by similar structures:

```
CONSTANT_FieldrefInfo {
    u1 tag;
    u2 class_index;
    u2 name_sig_index;
}
CONSTANT_MethodrefInfo {
    u1 tag;
    u2 class_index;
    u2 name_sig_index;
}
CONSTANT_InterfaceMethodrefInfo {
    u1 tag;
    u2 class_index;
    u2 name_sig_index;
}
```

The items of these structures are as follows:

tag

The tag item of a CONSTANT_FieldrefInfo structure has the value CONSTANT_Fieldref (9).

The tag item of a CONSTANT_MethodrefInfo structure has the value CONSTANT_Methodref (10).

The tag item of a CONSTANT_InterfaceMethodrefInfo structure has the value CONSTANT_InterfaceMethodref (11).

classs_index

The value of the class_index item must represent a valid Java class or interface name. The name represented by class_index must be exactly the same name that is described by the corresponding CONSTANT_Class_info entry in the constant_pool of the original class file.

name_sig_index

The value of the name_sig_index item must represent a valid Java name and type. The name and type represented by name_sig_index must be exactly the same name and type described by the CONSTANT_NameAndType_info entry in the constant_pool structure of the original class file.

Class

Each class is described by a fixed-length ClassInfo structure. The format of this structure is:

```
ClassInfo {
    u2 name_index;
    u1 max_field;
    u1 max_sfield;
    u1 max_method;
    u1 max_interface;
    u2 superclass;
    u2 access_flags;
```

```
    FieldInfo field[max_field+max_sfield];
    InterfaceInfo interface[max_interface];
    MethodInfo method[max_method];
}
```

The items of the ClassInfo structure are as follows:

name_index

The value of the name_index item must represent a valid Java class name. The Java class name represented by name_index must be exactly the same Java class name that is described in the corresponding ClassFile structure of the original class file.

max_field

The value of the max_field item gives the number of FieldInfo (0) structures in the field table that represent the instance variables, declared by this class or interface type. This value refers to the number of non-static the fields in the card class file. If the class represents an interface the value of max_field is 0.

max_sfield

The value of the max_sfield item gives the number of FieldInfo structures in the field table that represent the class variables, declared by this class or interface type. This value refers to the number of static the fields in the card class file.

max_method

The value of the max_method item gives the number of MethodInfo (0) structures in the method table.

max_interface

The value of the max_interface item gives the number of direct superinterfaces of this class or interface type.

superclass

For a class, the value of the superclass item must represent a valid Java class name. The Java class name represented by superclass must be exactly the same Java class name that is described in the corresponding ClassFile structure of the original class file. Neither the superclass nor any of its superclasses may be a final class.

If the value of superclass is 0[1], then this class must represent the class java.lang.Object, the only class or interface without a superclass.

For an interface, the value of superclass must always represent the Java class java.lang.Object.

access_flags

The value of the access_flags item is a mask of modifiers used with class and interface declarations. The access_flags modifiers and their values are the same as the access_flags modifiers in the corresponding ClassFile structure of the original class file.

field[]

Each value in the field table must be a fixed-length FieldInfo (0) structure giving a complete description of a field in the class or interface type. The field table includes only those fields that are declared by this class or interface. It does not include items representing fields that are inherited from superclasses or superinterfaces.

interface[]

Each value in the interface array must represent a valid interface name. The interface name represented by each entry must be exactly the same interface name that is described in the corresponding interface array of the original class file.

method[]

Each value in the method table must be a variable-length MethodInfo (0) structure giving a complete description of and Java Virtual Machine code for a method in the class or interface.

The MethodInfo structures represent all methods, both instance methods and, for classes, class (static) methods, declared by this class or interface type. The method table only includes those methods that are explicitly declared by this class. Interfaces have only the single method <clinit>, the interface initialization method. The methods table does not include items representing methods that are inherited from superclasses or superinterfaces.

---

[1] Or a standard yet fixed value.

Fields

Each field is described by a fixed-length field_info structure. The format of this structure is

```
FieldInfo {
    u2 name_index;
    u2 signature_index;
    u2 access_flags;
}
```

The items of the FieldInfo structure are as follows:

name_index

The value of the name_index item must represent a valid Java field name. The Java field name represented by name_index must be exactly the same Java field name that is described in the corresponding field_info structure of the original class file.

signature_index

The value of the signature_index item must represent a valid Java field descriptor. The Java field descriptor represented by signature index must be exactly the same Java field descriptor that is described in the corresponding field_info structure of the original class file.

access_flags

The value of the access_flags item is a mask of modifiers used to describe access permission to and properties of a field. The access_flags modifiers and their values are the same as the access_flags modifiers in the corresponding field_info structure of the original class file.

Methods

Each method is described by a variable-length MethodInfo structure. The MethodInfo structure is a variable-length structure that contains the Java Virtual Machine instructions and auxiliary information for a single Java method, instance initialization method, or class or interface initialization method. The structure has the following format:

```
MethodInfo {
    u2 name_index;
    u2 signature_index;
    u1 max_local;
    u1 max_arg;
    u1 max_stack;
    u1 access_flags;
    u2 code_length;
    u2 exception_length;
    u1 code[code_length];
    {     u2 start_pc;
          u2 end_pc;
          u2  handler_pc;
          u2  catch_type;
    } einfo[exception_length];
}
```

The items of the MethodInfo structure are as follows:

name_index

The value of the name_index item must represent either one of the special internal method names, either <init> or <clinit>, or a valid Java method name. The Java method name represented by name_index must be exactly the same Java method name that is described in the corresponding method_info structure of the original class file.

signature_index

The value of the signature_index item must represent a valid Java method descriptor. The Java method descriptor represented by signature_index must be exactly the same Java method descriptor that is described in the corresponding method_info structure of the original class file.

max_l cal

The value of the max_locals item gives the number of local variables used by this method, excluding the parameters passed to the method on invocation. The index of the first local variable is 0. The greatest local variable index for a one-word value is max_locals-1.

max_arg

The value of the max_arg item gives the maximum number of arguments to this method.

max_stack

The value of the max_stack item gives the maximum number of words on the operand stack at any point during execution of this method.

access_flags

The value of the access_flags item is a mask of modifiers used to describe access permission to and properties of a method or instance initialization method. . The access_flags modifiers and their values are the same as the access_flags modifiers in the corresponding method_info structure of the original class file.

code_length

The value of the code_length item gives the number of bytes in the code array for this method. The value of code_length must be greater than zero; the code array must not be empty.

exception_length

The value of the exception_length item gives the number of entries in the exception_info table.

code[]

The code array gives the actual bytes of Java Virtual Machine code that implement the method. When the code array is read into memory on a byte addressable machine, if the first byte of the array is aligned on a 4-byte boundary, the tableswitch and lookupswitch 32-bit offsets will be 4-byte aligned; refer to the descriptions of those instructions for more information on the consequences of code array alignment.

The detailed constraints on the contents of the code array are extensive and are the same as described in the Java Virtual Machine Specification.

einfo[]

Each entry in the einfo array describes one exception handler in the code array. Each einfo entry contains the following items:

start_pc, end_pc

The values of the two items start_pc and end_pc indicate the ranges in the code array at which the exception handler is active.

The value of start_pc must be a valid index into the code array of the opcode of an instruction. The value of end_pc either must be a valid index into the code array of the opcode of an instruction, or must be equal to code_length, the length of the code array. The value of start_pc must be less than the value of end_pc.

The start_pc is inclusive and end_pc is exclusive; that is, the exception handler must be active while the program counter is within the interval [start_pc, end_pc].

handler_pc

The value of the handler_pc item indicates the start of the exception handler. The value of the item must be a valid index into the code array, must be the index of the opcode of an instruction, and must be less than the value of the code_length item.

catch_type

If the value of the catch_type item is nonzero, it must represent a valid Java class type. The Java class type represented by catch_type must be exactly the same as the Java class type that is described by the catch_type in the corresponding method_info structure of the original class file. This class must be the class Throwable or one of its subclasses. The exception handler will be called only if the thrown exception is an instance of the given class or one of its subclasses.

If the value of the catch_type item is zero, this exception handler is called for all exceptions. This is used to implement finally.

Attributes

Attributes used in the original class file are either eliminated or regrouped for compaction.

The predefined attributes SourceFile, ConstantValue, Exceptions, LineNumberTable, and Local-VariableTable may be eliminated without sacrificing any information required for Java byte code interpretation.

The predefined attribute Code which contains all the byte codes for a particular method are moved in the corresponding MethodInfo structure.

Constraints on Java Card Virtual Machine Code

The Java Card Virtual Machine code for a method, instance initialization method, or class or interface initialization method is stored in the array code of the MethodInfo structure of a card class file. Both the static and the structural constraints on this code array are the same as those described in the Red book.
Limitations of the Java Card Virtual Machine and Java Card class File Format
The following limitations in the Java Card Virtual Machine are imposed by this version of the Java Card Virtual Machine specification:

- The per-card class file constant pool is limited to 65535 entries by the 16-bit const_size field of the CardClassFile structure (0). This acts as an internal limit on the total complexity of a single card class file. This count also includes the entries corresponding to the constant pool of the class hierarchy available to the application in the card.[2]
- The amount of code per method is limited to 65535 bytes by the sizes of the indices in the MethodInfo structure.
- The number of local variables in a method is limited to 255 by the size of the max_local item of the MethodInfo structure (0).
- The number of fields of a class is limited to 510 by the size of the max_field and the max_sfield items of the ClassInfo structure (0).
- The number of methods of a class is limited to 255 by the size of the max_method item of the ClassInfo structure (0).
- The size of an operand stack is limited to 255 words by the max_stack field of the MethodInfo structure (0).

Bibliography

[1] Tim Lindholm and Frank Yellin, The Java Virtual Machine Specification, Addison-Wesley, 1996.

[2] A single card class file constant pool has 65535-Δ entries available, where Δ corresponds to the number of entries in the constant pool of the class hierarchies accessible to the application.